Design Documentation

CSE 453: Northrop Grumman Bird Chirp Classifier

Table of Contents

Project Description

Team Members

Justin Charlong, jrcharlo@buffalo.edu
Aditya Singh Rathore, asrathor@buffalo.edu
Jared Payne, jaredpay@buffalo.edu
Kevin Jiang, kevinjia@buffalo.edu

Project GIT Repository

https://www.github.com/asrathor/BirdChirper

Project Objective

The goal is to develop a system that identifies bird types from the sound of their chirps. The system processes the audio input data to recognize the type of bird that made the chirp sound. We acquired audio data from a collection of North American birds and performed feature extraction for computing a set of features for each bird. The computed features are fed to the Machine Learning Algorithm (Neural Network) for identification of birds. Both feature extraction and machine learning is implemented on GPU for high performance.

Data Collection

For the evaluation of our identification model, 8 North American birds are chosen that varies from each other on wide range of frequencies. The sample data for the birds is acquired from www.xeno-canto.org, which is an online database for sharing wide variety of bird sounds. The birds that are chosen for our project are shown below:

- American Crow
- American Kestrel
- American Yellow Warbler
- Blue Jay
- Canada Goose
- Grey Catbird
- Green Winged Teal
- Northern Cardinal

System Setup

For achieving the best performance, the feature extraction and machine learning algorithm is implemented on the GPU platform. The setup process can be categorized in three steps:

1. libsndfile1-dev: This library is used for reading the input WAV files.
2. Nvidia CUDA Toolkit v8.0: The toolkit is used for computing Fast Fourier Transforms during feature extraction.
3. TensorFlow for GPU: This library is used to perform Machine Learning computations.

Feature Extraction

Data Formatting

Before computing features, the acquired bird sounds are converted from MP3 to mono WAV format. It is necessary to exclude the noise from the WAV files to avoid discrepancies in features. One such tool that can be used for noise removal is Audacity software, which provides functions for noise calculation and noise reduction. In order to get accurate features only the bird chirps are taken as inputs, rather than the whole signal which contains many stagnant signals.

Functions

The computation of features is implemented using C++ on GPU platform. The C++ code, namely extractFeatures.cu is located at CUDA/extractFeatures. To compile the executable, use the makefile referenced from NVIDIA (see reference for link). In the program the type Complex is an alias for the type float2, defined by the CUDA libraries. The code comprises of three functions whose descriptions are as follows:

main

Input: int argc, char** argv
Output: 0 on success, error otherwise
This function is used to first parse the command line arguments and acquire the data files in WAV format. The feature computations can be performed a single WAV file (argc = 2) or a batch containing all the WAV files for a specific bird (argc = 3). Depending on the argc parameters, the respective number of files are taken as input and are provided as arguments for extractFeatures function. If more than required command line arguments are used, then the correct form of providing the input is outputted to the user.

extractFeatures

Input: int argc, char** argv, const* char filename
Output: void
This function is used for performing calculations of seven features for a sample WAV file. First, information about number of frames, sample rate and channels are computed for the sample file. Using these information, respective memory for reading the data and host memory for signal is assigned. It is necessary to convert the values of sample data from int to Complex since CUDA libraries accept the inputs in Complex format. After the conversion, the Fast Fourier Transform (FFT) is computed on modified sample data using CUDA cuFFT library. Most of the features are calculated by using different frames of FFT. Therefore, the FFT is segmented into multiple frames with frame length of 2 ms. Following are the seven features that utilizes the FFT for computations:

Name	Description	Equation
Spectral Centroid	The center of mass of the energy spectrum of a signal.	$c_i = \frac{\sum_{u=0}^{M} u.\lvert f_i(u)\rvert^2}{\sum_{u=0}^{M}\lvert f_i(u)\rvert^2}$
Bandwidth	The range of frequencies for which there is relevant spectral energy for the signal.	$b_i^2 = \frac{\sum_{u=0}^{M}(u-c_i)^2.\lvert f_i(u)\rvert^2}{\sum_{u=0}^{M}\lvert f_i(u)\rvert^2}$
Spectral Rolloff Frequency	The frequency at which a certain percentage of the total spectral energy of a signal is contained below (for our calculations, we are currently finding 92% of total energy).	$SRF = TH\sum_{n=0}^{M}\lvert X(n)\rvert^2$ where TH = 0.92
Spectral Flatness	Provides a figure (in dB) which corresponds to how noise-like a signal is.	$SF = 10log_{10}(\frac{\sqrt[1/M]{\prod_{i=0}^{M}\lvert X_i\rvert}}{\sum_{i=0}^{M}\lvert X_i\rvert/_M})$
Zero Crossing Rate	Number of times the signal (as encoded in the WAV file) crosses zero divided by the number of samples.	$ZCR = \frac{1}{T-1}\sum_{t=1}^{T-1}1_{R<0}(s_t s_{t-1})$
Minimum and Maximum Frequencies	Minimum and maximum frequencies at -5 dB.	$X_{idB} = \log_{10}(X_i/X_{max})$

After the computations, the calculated features are written to a CSV format file to be later used by machine learning algorithm. The training labels are created for all the birds which will be used as an input to machine learning for verifying the identification model. At the end, the memory acquired by original bird signal and FFT is deallocated.

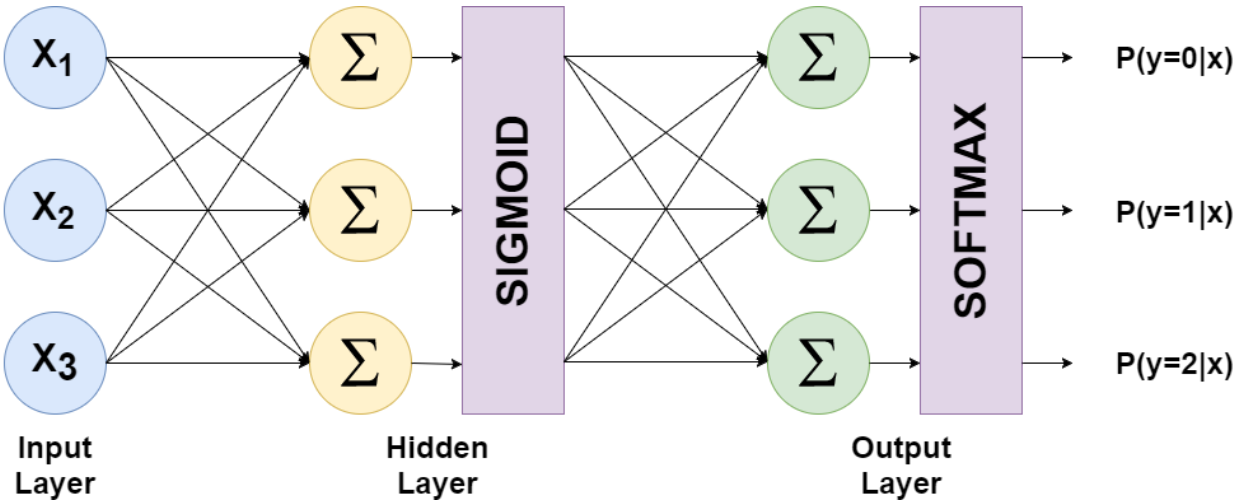
absComplex

Input: Complex n
Output: float
This function returns the absolute value of the Complex number n, which is computed as the square root of the sum of the square of the real part of n and the square of the complex part of the Complex n.

Adding Birds to the System

Due to the very scalable nature of the program, it is very easy to add birds to the system. The only part of the feature extraction code which needs to be edited to add a bird is the section which prints out the bird's labels inside of the extractFeatures function. To add birds to the system, an acronym must first be chosen by which all input files for that type of bird will include. Once an acronym has been chosen, an else if statement must be added under both of birds the "LABELFILE statements" comments (one for each mode of running the program). The number of 1's and 0's for each line correspond to the number of birds in the system. When adding a line for a new bird it is also important to increase the size of the new line as well as the previous lines. There should only be one entry in each line which is a 1.

Neural Network



Model Description

The general layout of the neural network is that it is composed of an input layer, single hidden layer, and output layer. The input layer is a 2D array of shape (Number of training samples, Number of features extracted). The hidden layer is where the 2D array from the input layer is multiplied by a 2D array of weight values of shape (Number of features extracted, Number of hidden nodes) where the output is a 2D array of shape (Number of training samples, Number of hidden nodes). The sigmoid function is applied to the 2D array and is then multiplied by a second 2D array of weight values of shape (Number of hidden layers, Number of unique labels) in the output layer which returns a 2D array of shape (Number of training samples, Number of unique labels). The softmax function is applied to the output and the result is the final step of the forward propagation. For each column the index with the highest probability value is the selected as the predicted bird species. The weights are then updated, based off the error, during backpropagation using Adam Optimization.

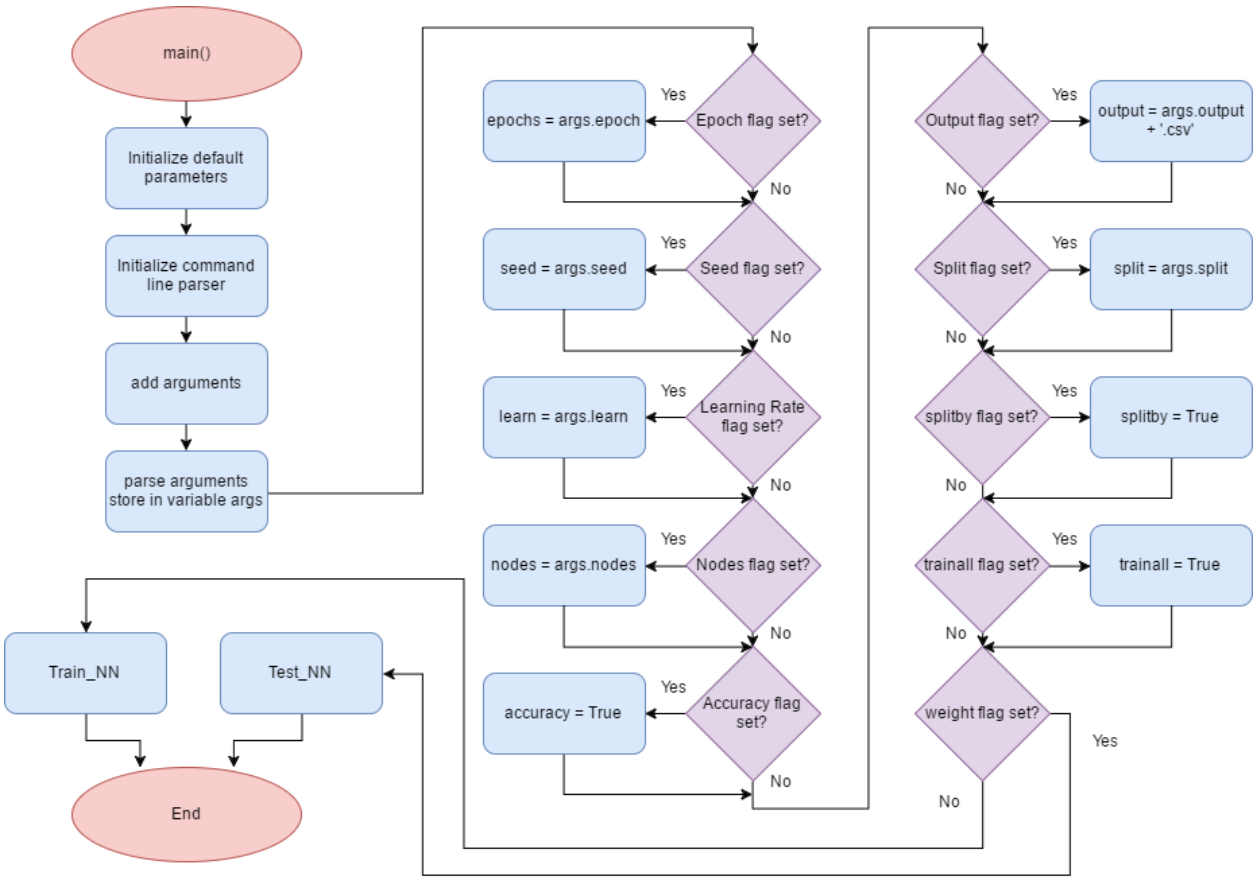
Functions

main

main()

Description:

The main function initializes the default parameters, parses all the arguments from the command line when the script is run, updates parameters based off of arguments passed in, and then either trains the neural network model or uses weights given by the user to make predictions.



import_data

import_data(datafile, labelfile, seed, split_by_train=True, split=10, split_data=True)

Description:

Imports the data from the given filenames and splits it accordingly into training and testing data sets.

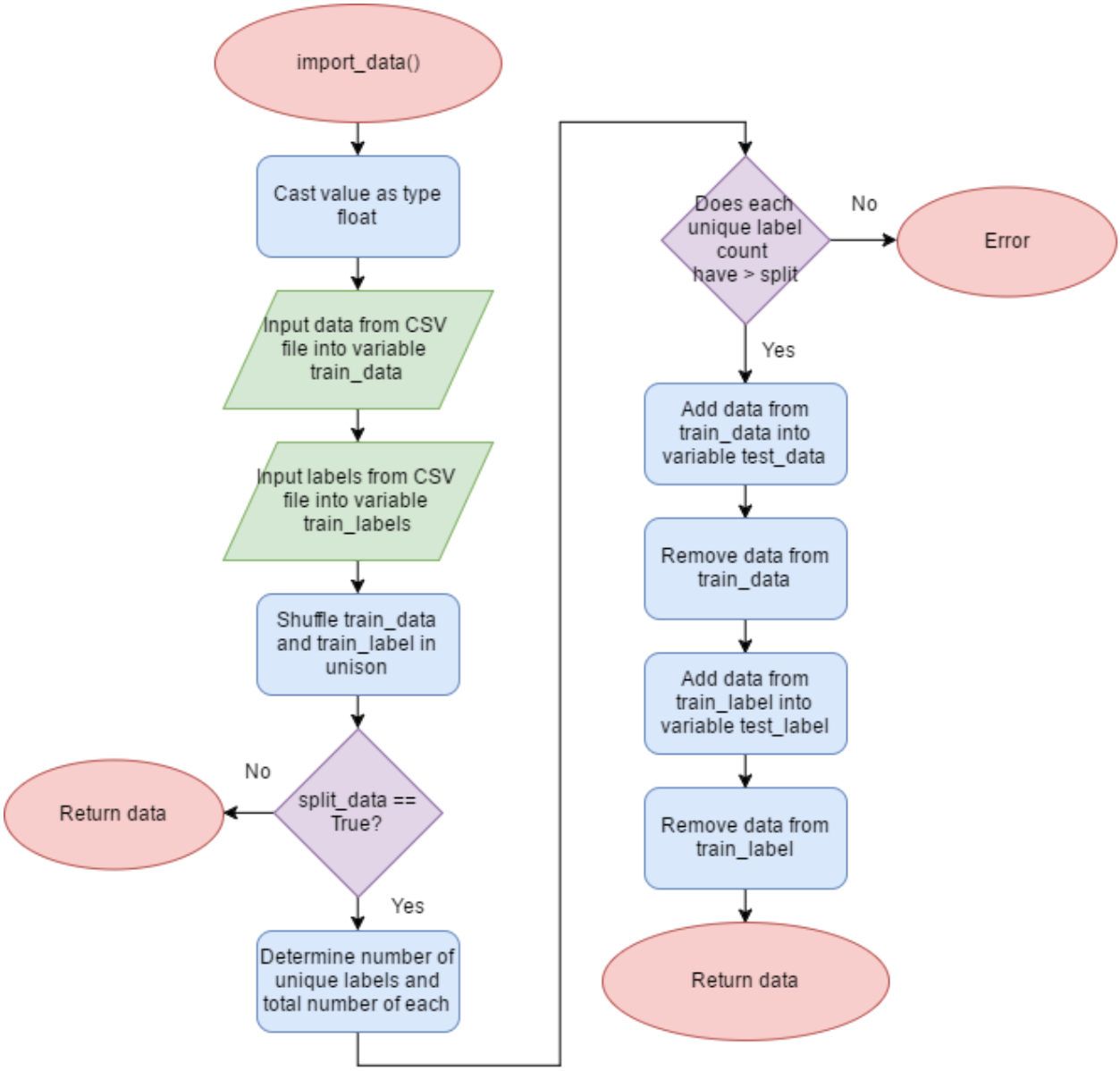
Returns the training and testing data and label sets as arrays.

Arguments:

- datafile : string
  The filename containing the training data and/or testing data.
- labelfile : string
  The filename containing the training label and/or testing label.
- seed : int
  A random number used for random shuffling.
- split_by_train : bool *optional*
  Default to True
  If True, split the inputted data by split as the training data.
  If False, split the inputted data by split as the testing data.
- split : int *optional*
  Default to 10
  The number of data to be used as training data or testing data.
- split_data : bool *optional*
  Default to True
  If True, split the inputted data into a training and testing data set.
  If False, use all inputted data as training data.

Returns:

- train_data : 2D array
  The array of training data
- test_data : 2D array *optional*
  Only provided if split_data is True
  The array of testing data
- train_label : column array
  The array of training true labels
- test_label : column array *optional*
  Only provided if split_data is True
  The array of testing true labels


import_data()
Cast value as type float
Input data from CSV file into variable train_data
Input labels from CSV file into variable train_labels
Shuffle train_data and train_label in unison
split_data == True?
No
Return data
Yes
Determine number of unique labels and total number of each
Does each unique label count have > split
No
Error
Yes
Add data from train_data into variable test_data
Remove data from train_data
Add data from train_label into variable test_label
Remove data from train_label
Return data

init_weights

init_weights(shape)

Description:

Initialize starting weights for a new Neural Network.

Arguments:

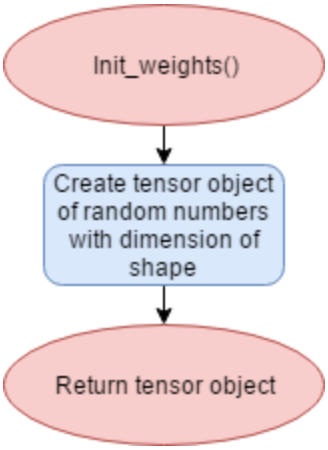
- shape : int

  The dimension of the array of weights.

Return:

- weight : Tensorflow object

  A Tensorflow object containing the initial weights.



forwardprop

forwardprop(X, w_1, w_2)

Description:

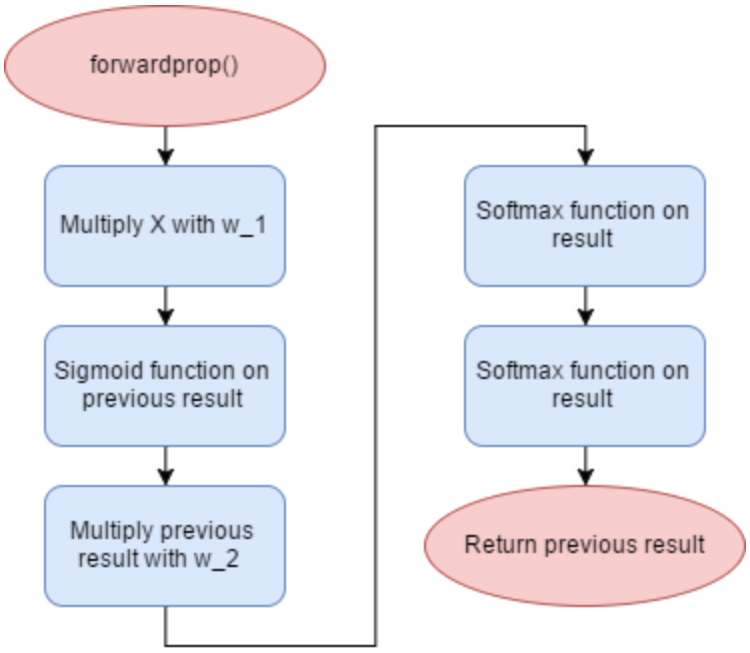
Perform the feedforward propagation of using sigmoid as the activation function for the hidden layer and softmax function for the output layer.

Arguments:

- X : Tensorflow placeholder
  A placeholder where the input data will be in.
- w_1 : Tensorflow object
  A Tensorflow object containing the hidden layer weights.
- w_2 : Tensorflow object
  A Tensorflow object containing the output layer weights.

Return:

- y_out : Tensorflow object
  A Tensorflow object containing the outputs of each output layer node's value.



Init_existing_weights

Init_existing_weights(file)

Description:

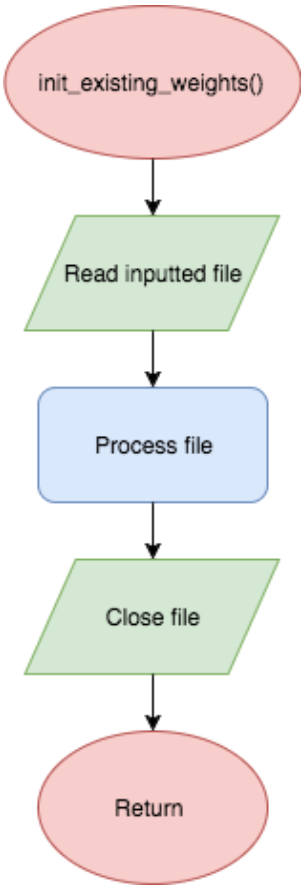
Imports existing weights from the given filename.

Arguments:

- file : string
  The filename containing weights.

Returns:

- x_size : int
  The number of input layer nodes.
- h_size : int
  The number of hidden layer nodes.
- y_size : int
  The number of output layer nodes.
- w_1 : Tensorflow object
  A Tensorflow object containing the weights from input layer to hidden layer.
- w_2 : Tensorflow object
  A Tensorflow object containing the weights from hidden layer to output layer.



Train_NN

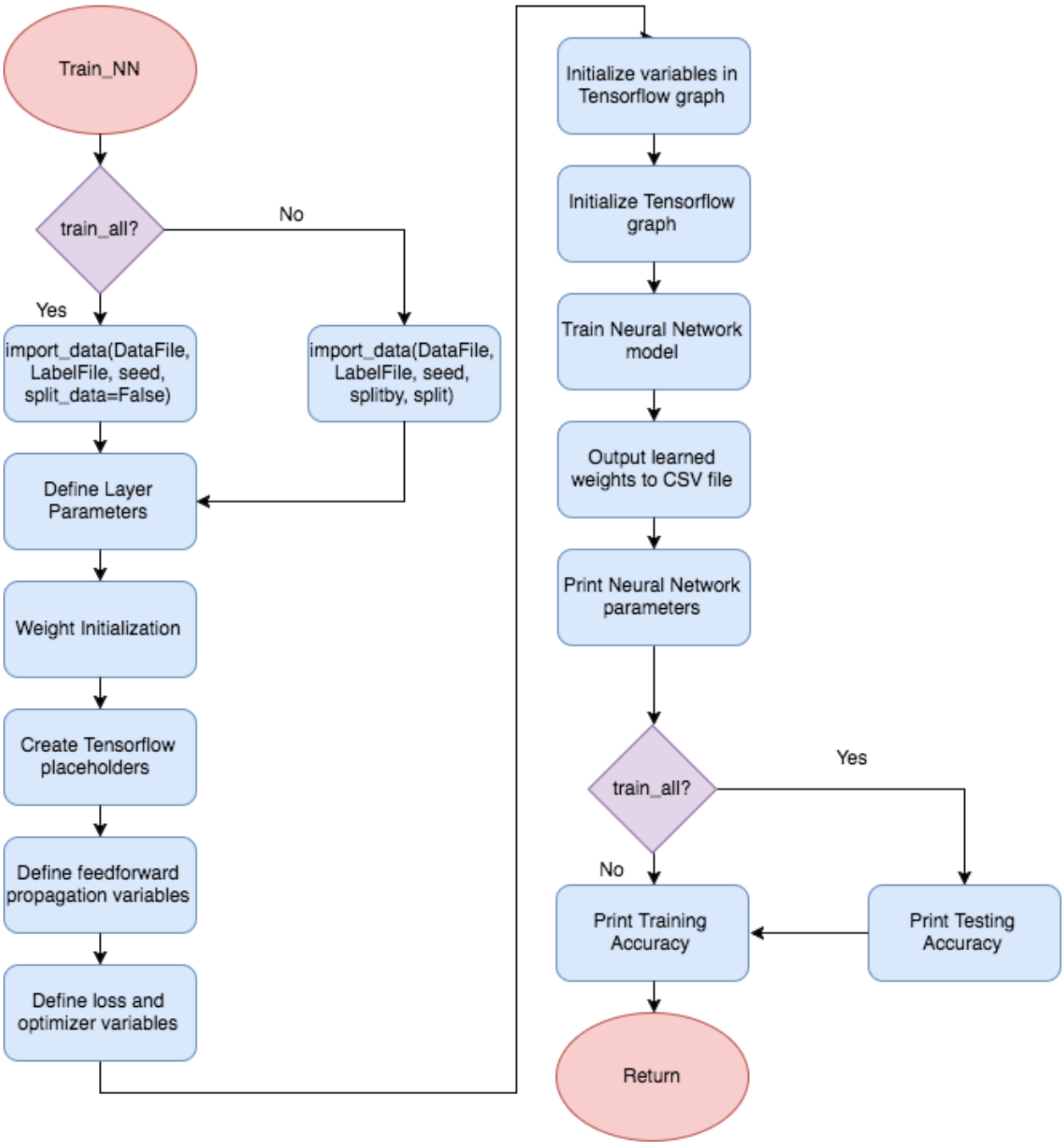
Train_NN(max_epochs, n_hidden, DataFile, LabelFile, seed, learning_rate, show_accuracy, output_file, splitby, split, train_all)

Description:

Trains a new Neural Network with the provided parameters.

Arguments:

- max_epochs : int
  The number of iterations to train.
- n_hidden : int
  The number of hidden layer nodes.
- DataFile : string
  The training data filename.
- LabelFile : string
  The training label filename.
- seed : int
  A random number used for random shuffling.
- learning_rate : float
  The learning rate of the Neural Network.
- show_accuracy : bool
  If True, show the accuracy of the Neural Network during every iteration of the training step.
  If False, hide the mentioned accuracy.
- output_file : string
  The name of the output file containing the weights.
- splitby : bool
  If True, split the inputted data by split as the training data.
  If False, split the inputted data by split as the testing data.
- split : int
  The number of data to split the inputted data by.
- train_all : bool
  If True, use all the inputted data as the training data.


Train_NN
train_all?
No
Yes
import_data(DataFile, LabelFile, seed, split_data=False)
import_data(DataFile, LabelFile, seed, splitby, split)
Define Layer Parameters
Weight Initialization
Create Tensorflow placeholders
Define feedforward propagation variables
Define loss and optimizer variables
Initialize variables in Tensorflow graph
Initialize Tensorflow graph
Train Neural Network model
Output learned weights to CSV file
Print Neural Network parameters
train_all?
Yes
No
Print Training Accuracy
Print Testing Accuracy
Return

Test_NN

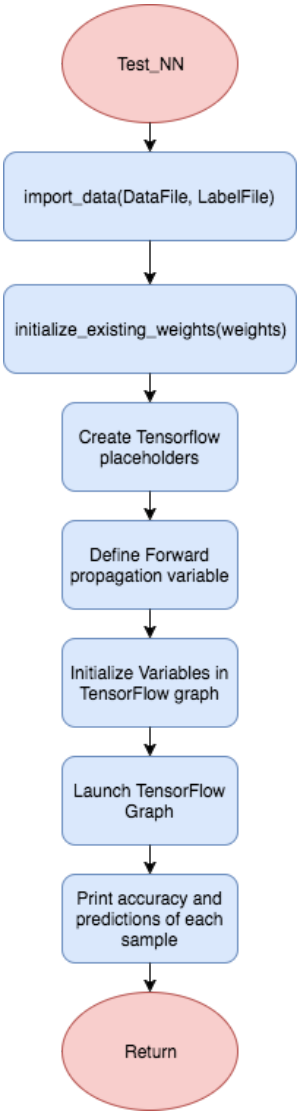
Test_NN(weights, DataFile, LabelFile, seed)

Description:

Predicts the label of the provided data using the given weights.

Arguments:

- weights : string
  The filename containing the weights of the Neural Network.
- DataFile : string
  The filename containing the testing data.
- LabelFile : string
  The filename containing the testing labels.



check_positive_int

check_positive_int (value)

Description:

Convert string representation of an integer number to type int and check if the value is greater than 0, else an error will be thrown.

Arguments:

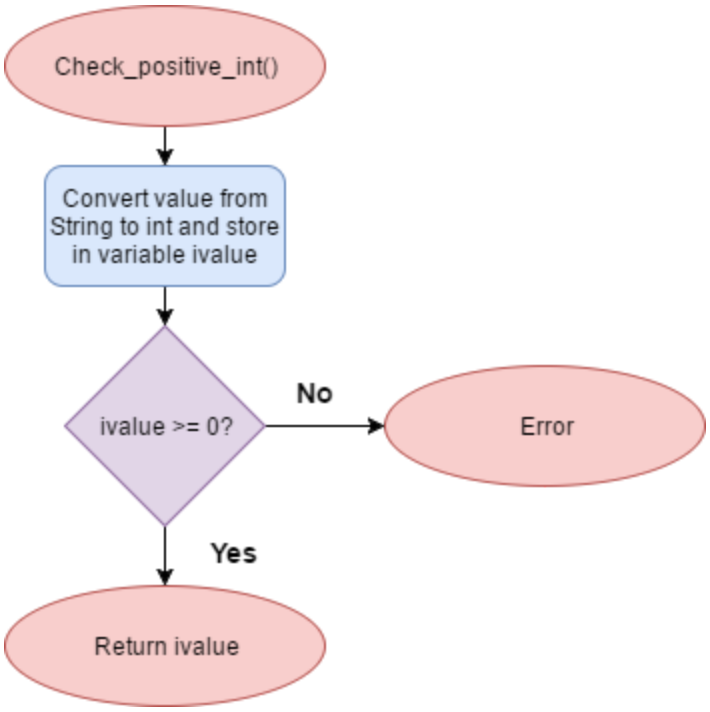
- value : String

  String representation of an integer number.

Returns:

- ivalue : int

  The integer value of the input string.



check_positive_float

check_positive_float (value)

Description:

Convert string representation of a float number to type float and check if the value is greater than 0, else an error will be thrown.

Arguments:

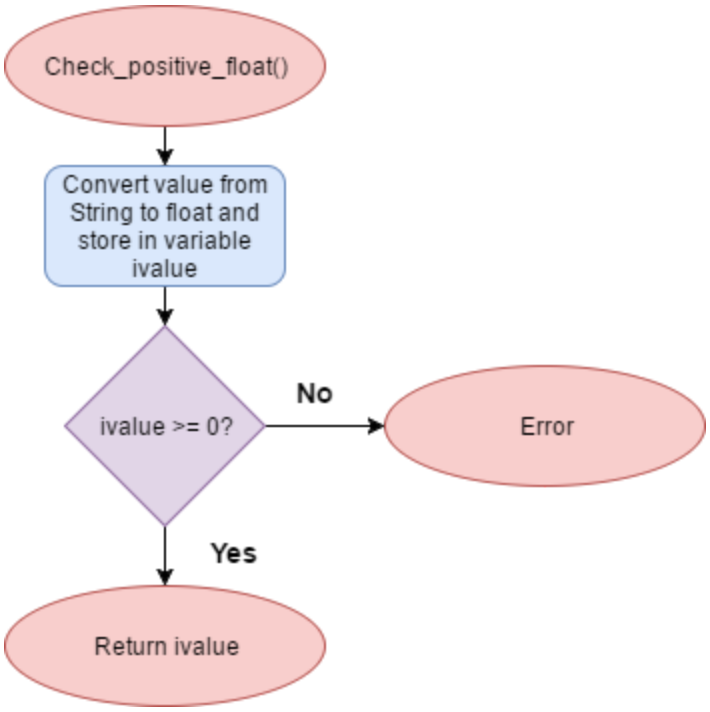
- value : String

  String representation of a float number.

Returns:

- ivalue : float

  The float value of the input string.



References

- Khuc, Vinh (2015) *Simple Feedforward Neural Networking using TensorFlow* [Source Code]. https://gist.github.com/vinhkhuc/e53a70f9e5c3f55852b0
- "CUDA Code Samples." NVIDIA Developer. N.p., 25 Apr. 2017. Web. 15 May 2017. http://developer.nvidia.com/cuda-code-samples